Searching & Hash Tables

CS1812/13: Object Oriented Programming II
Dr Reuben Rowe and Dr Matteo Sammartino
(based on slides by Dr Johannes Kinder)

Searching

- Many data structures are good at adding and removing data
- How to *search* for a particular value?
  - Decide whether an element is contained in a data structure

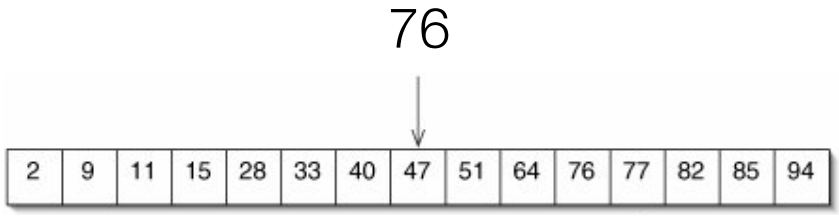
Linear Search

- Compare against each element
- If array has *n* elements, the algorithm needs on the order of *n* steps: the algorithm is in ***O(n)***, it is ***linear***

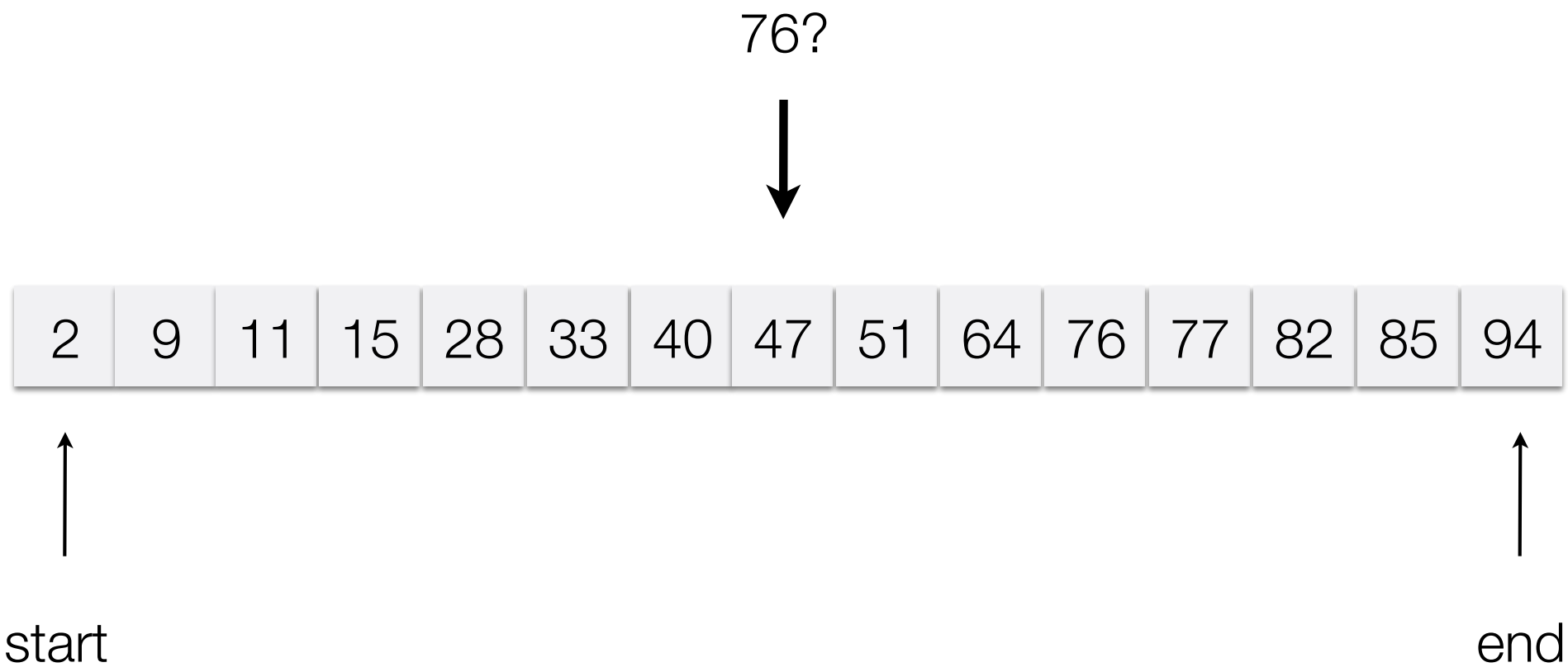
```
static boolean linearSearch(int[] arr, int val) {
  for (int x : arr) {
    if (x == val) {
      return true;
    }
  }
  return false;
}
```

Binary Search

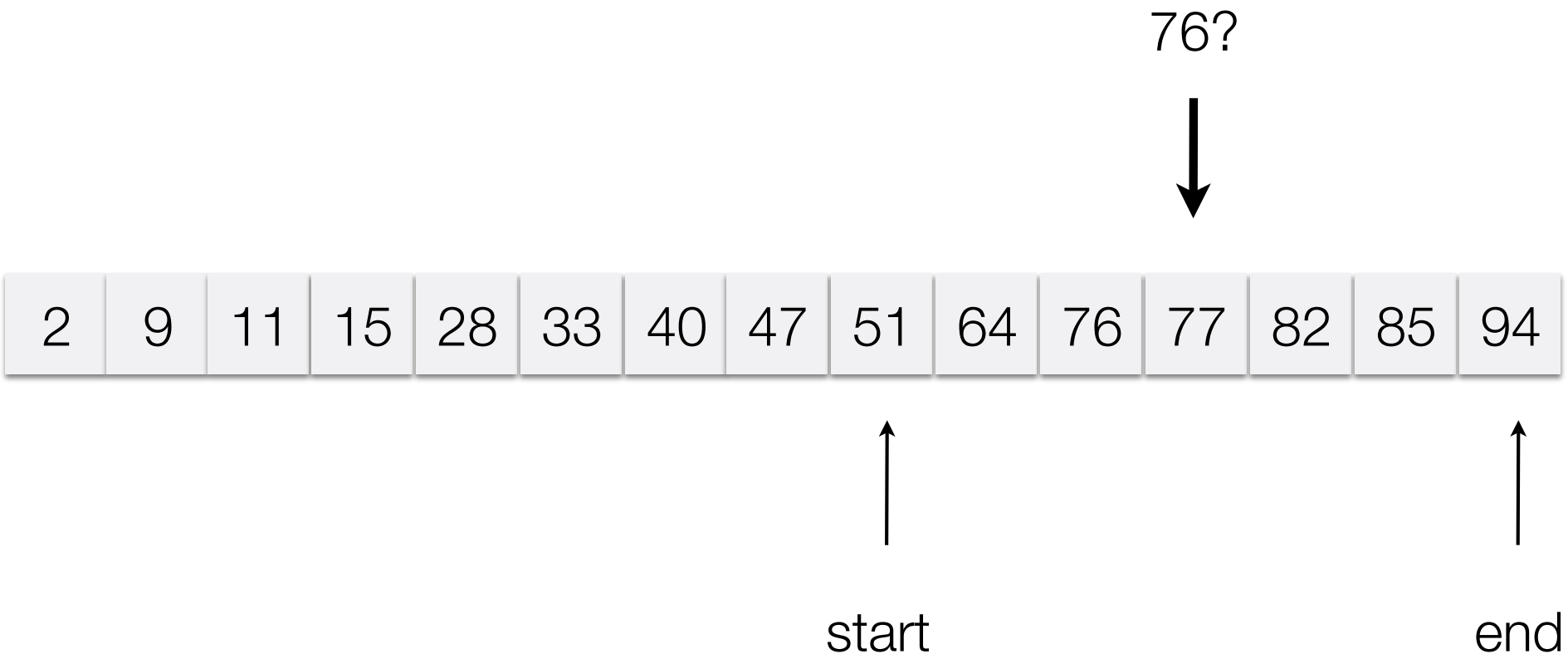
- What if the array is sorted?
- Idea
  - Compare against middle element → value has to be left or right
  - Recursively search in left or right half of the array



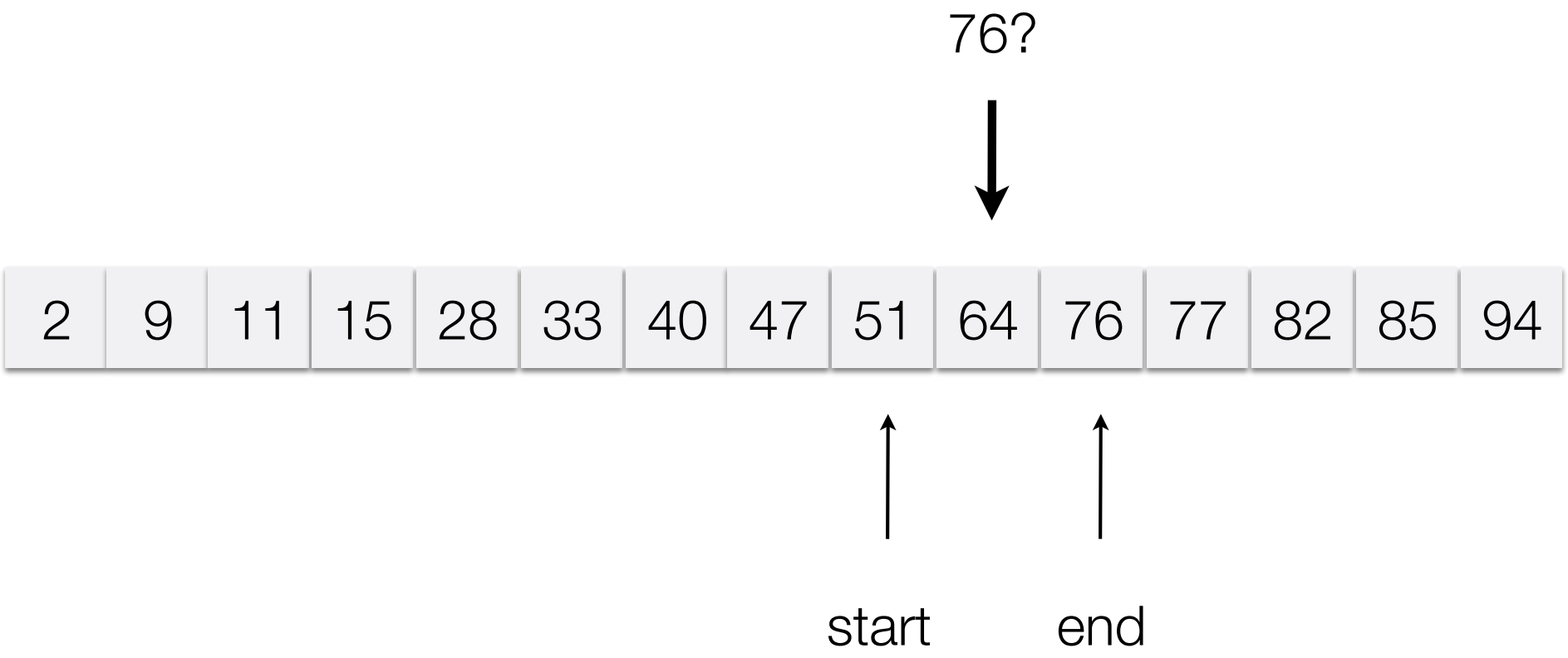
Binary Search



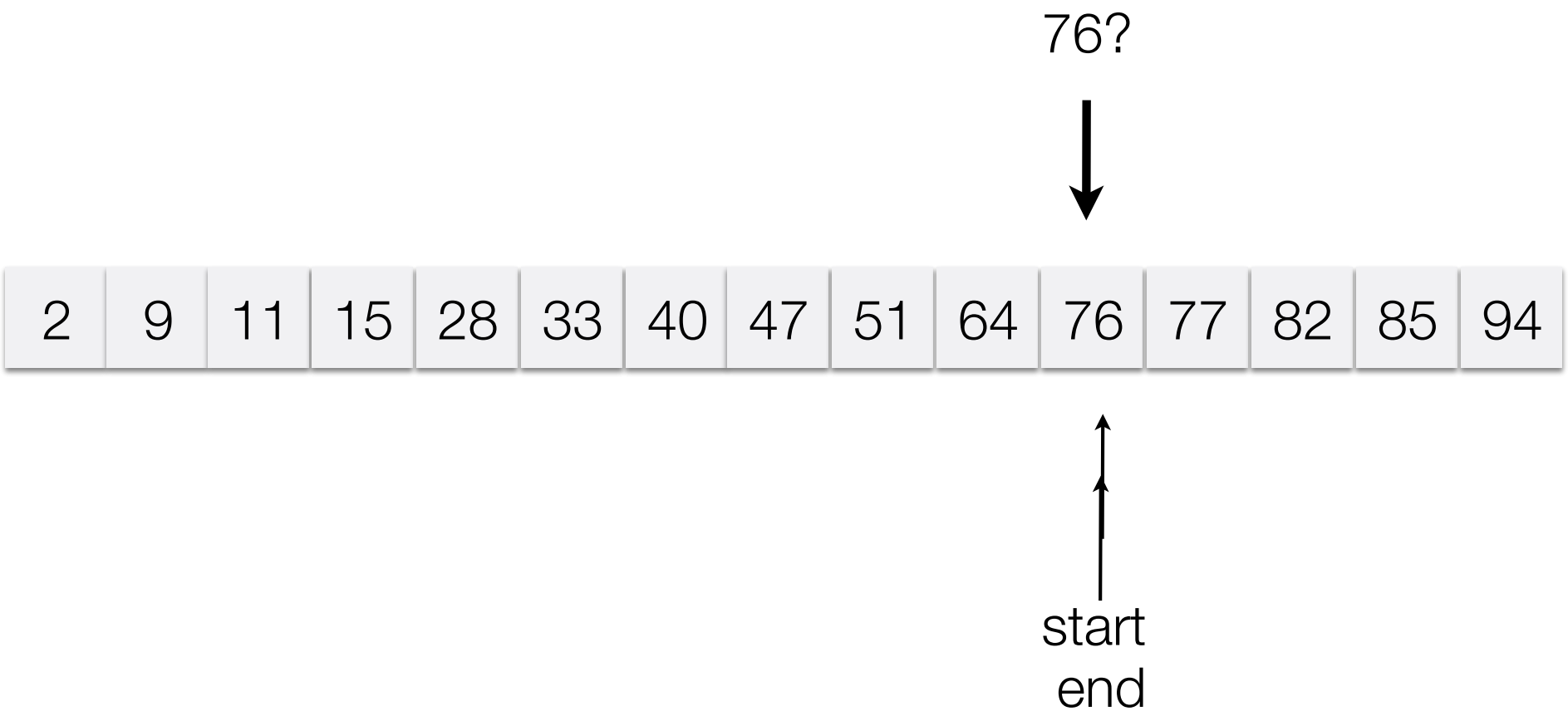
Binary Search



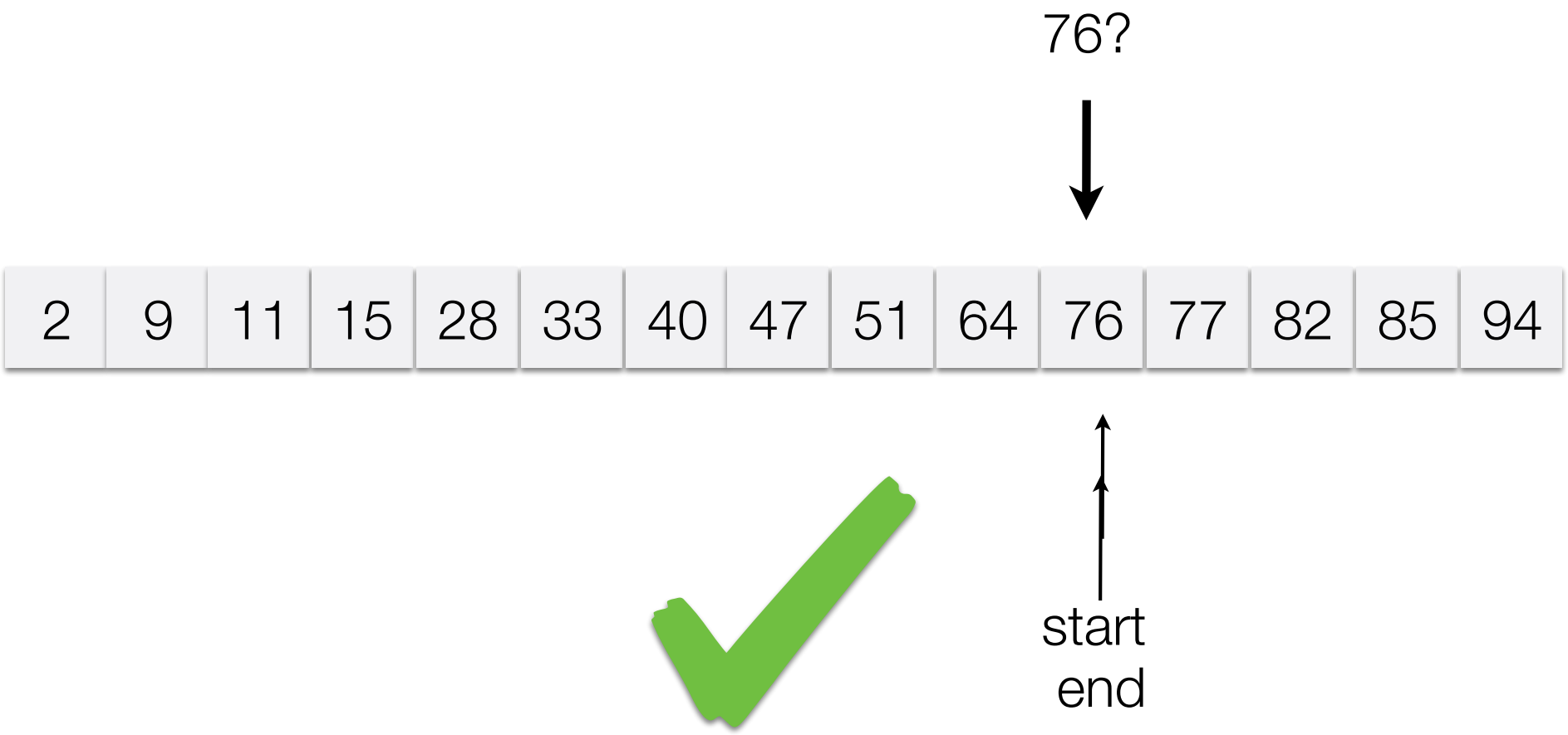
Binary Search



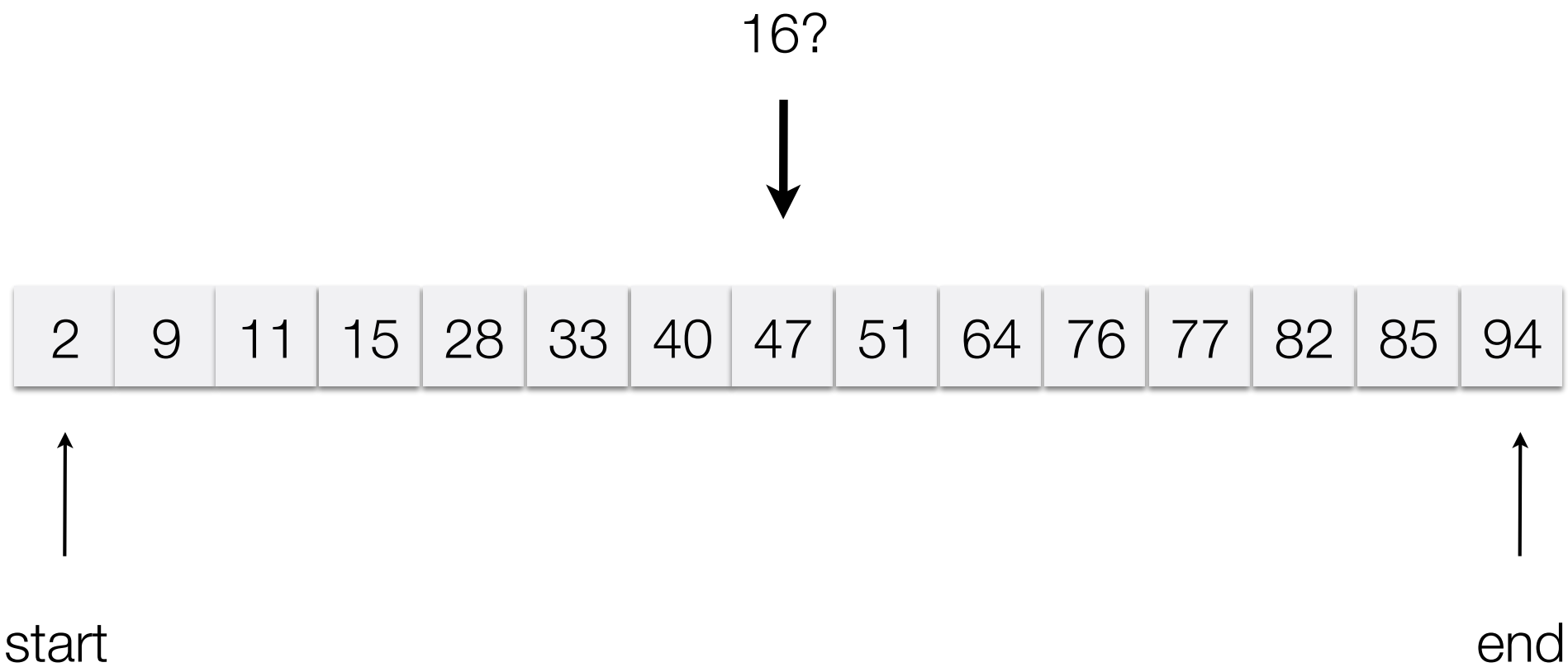
Binary Search



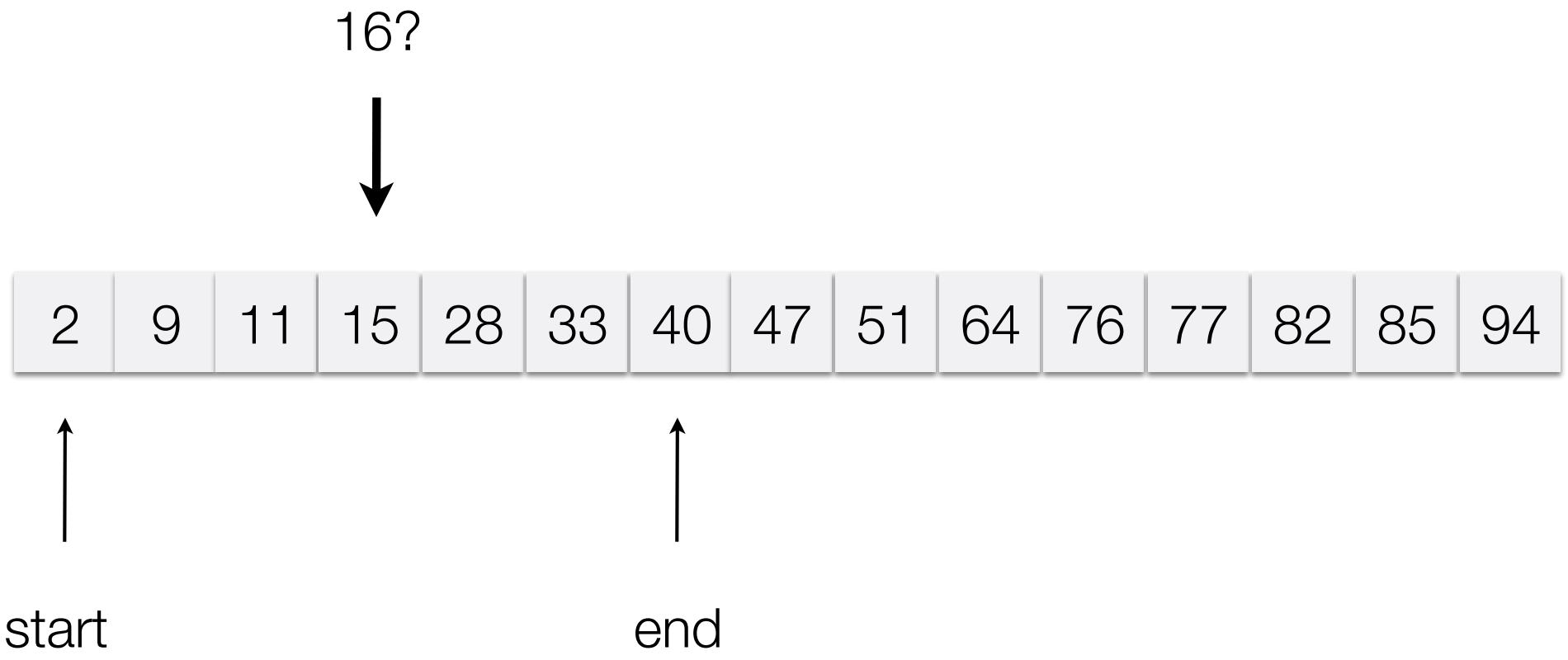
Binary Search



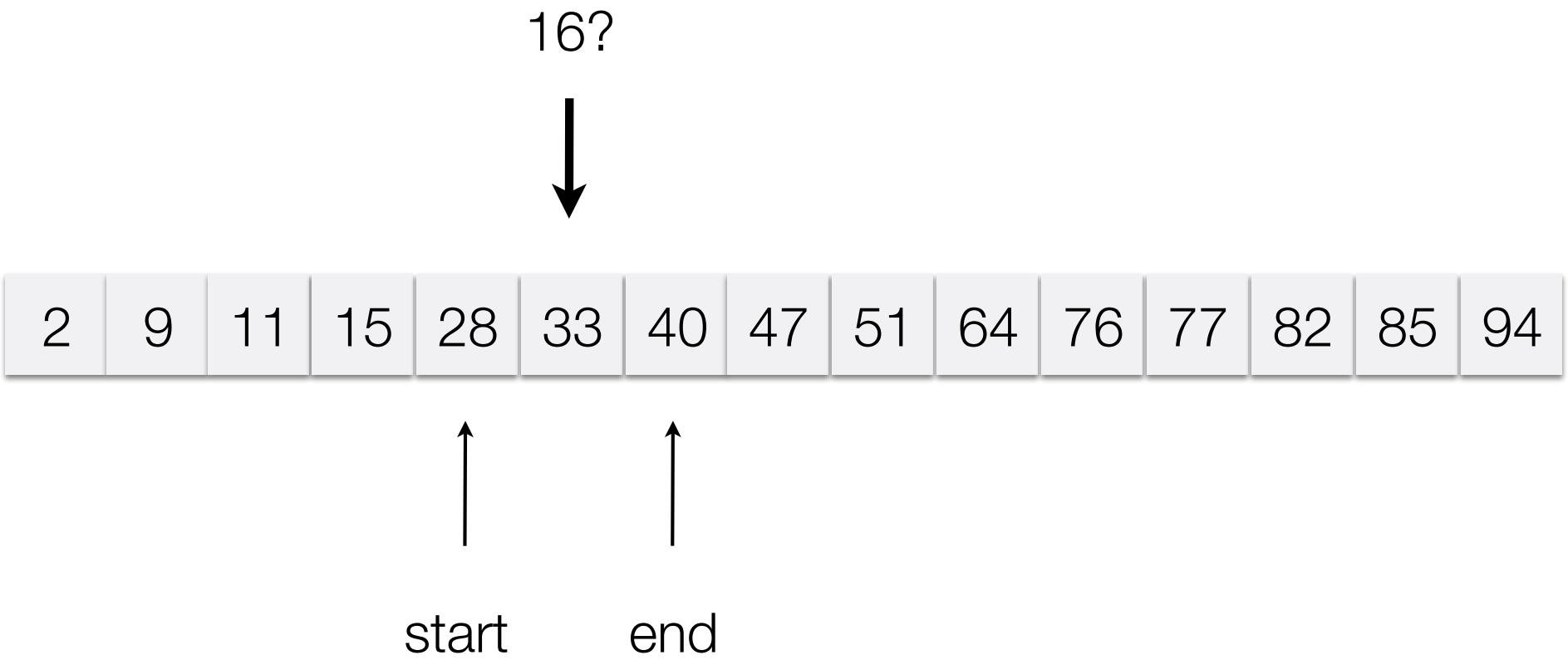
Binary Search



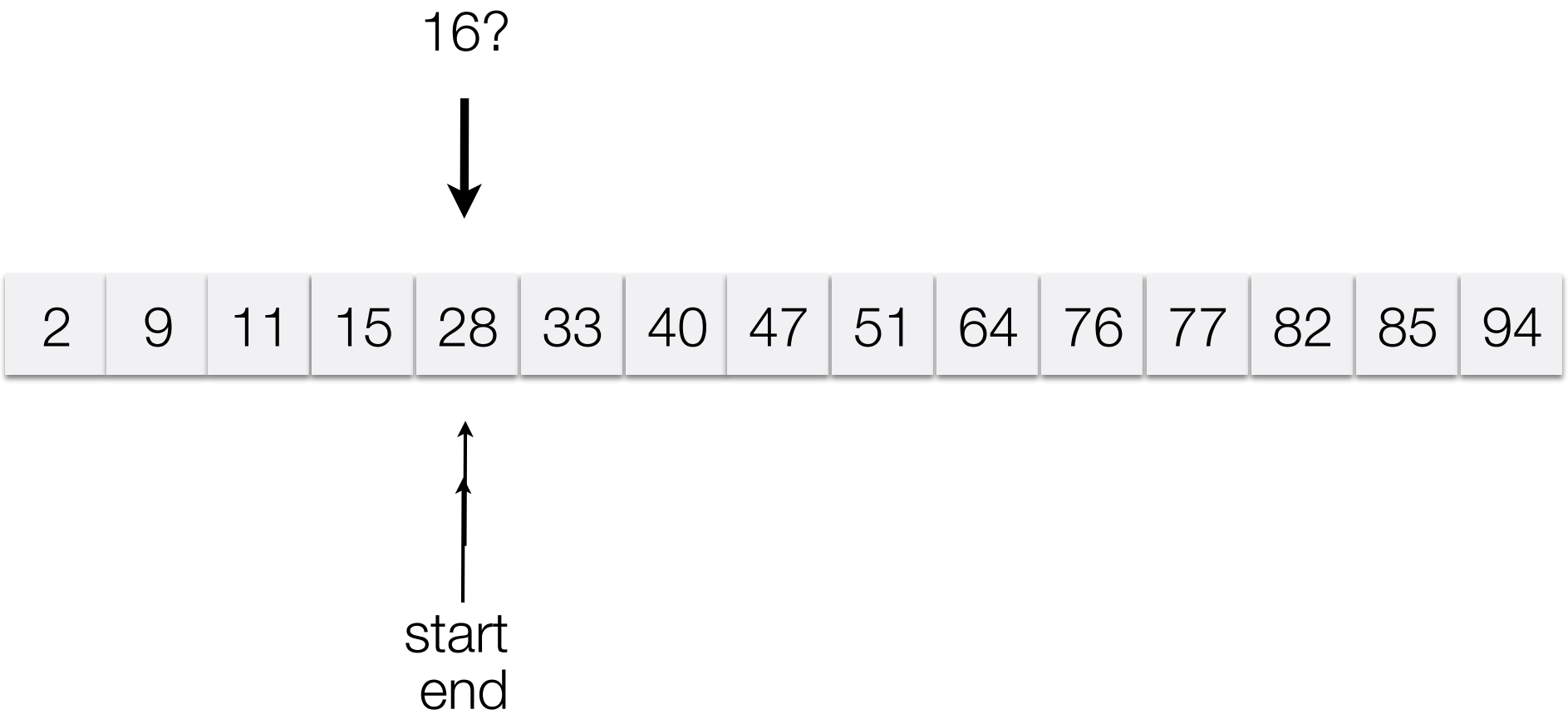
Binary Search



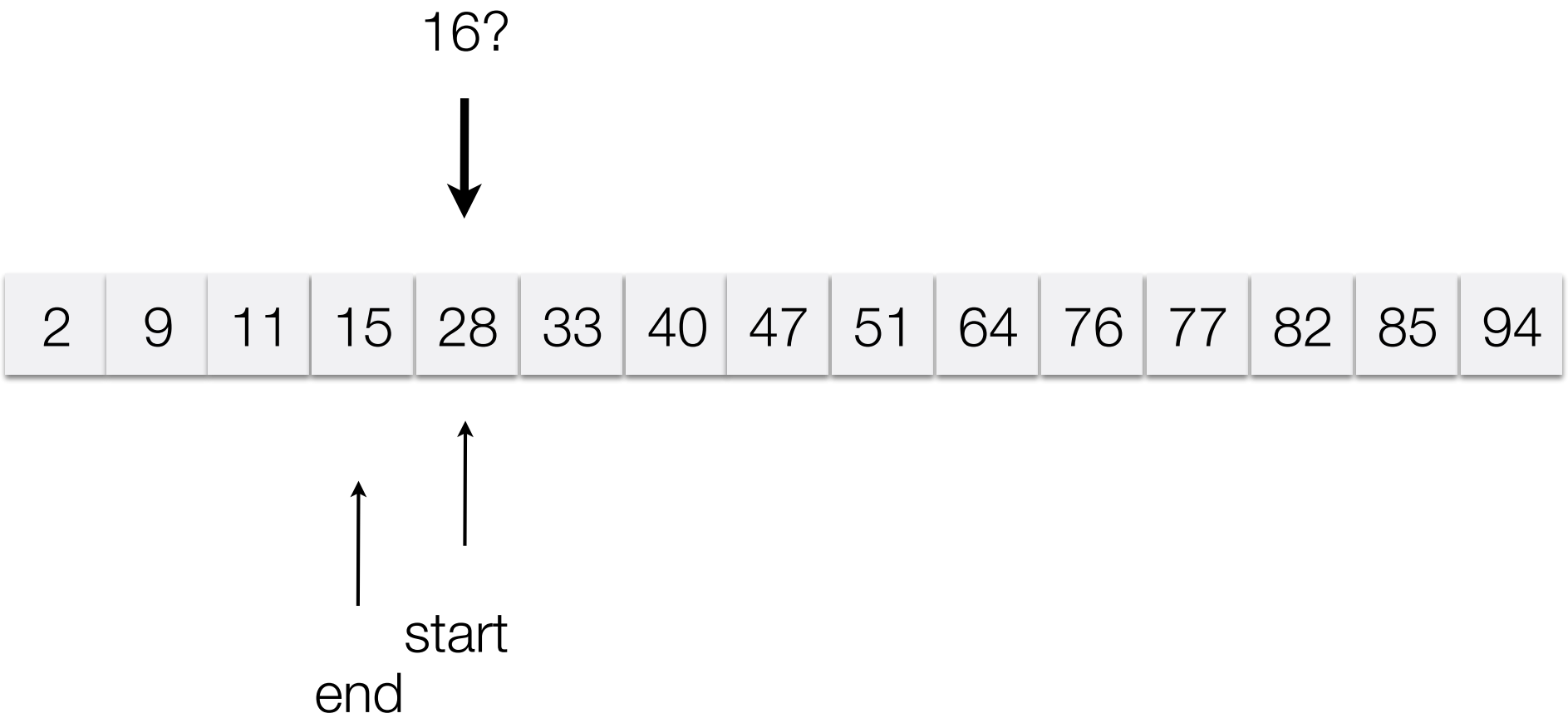
Binary Search



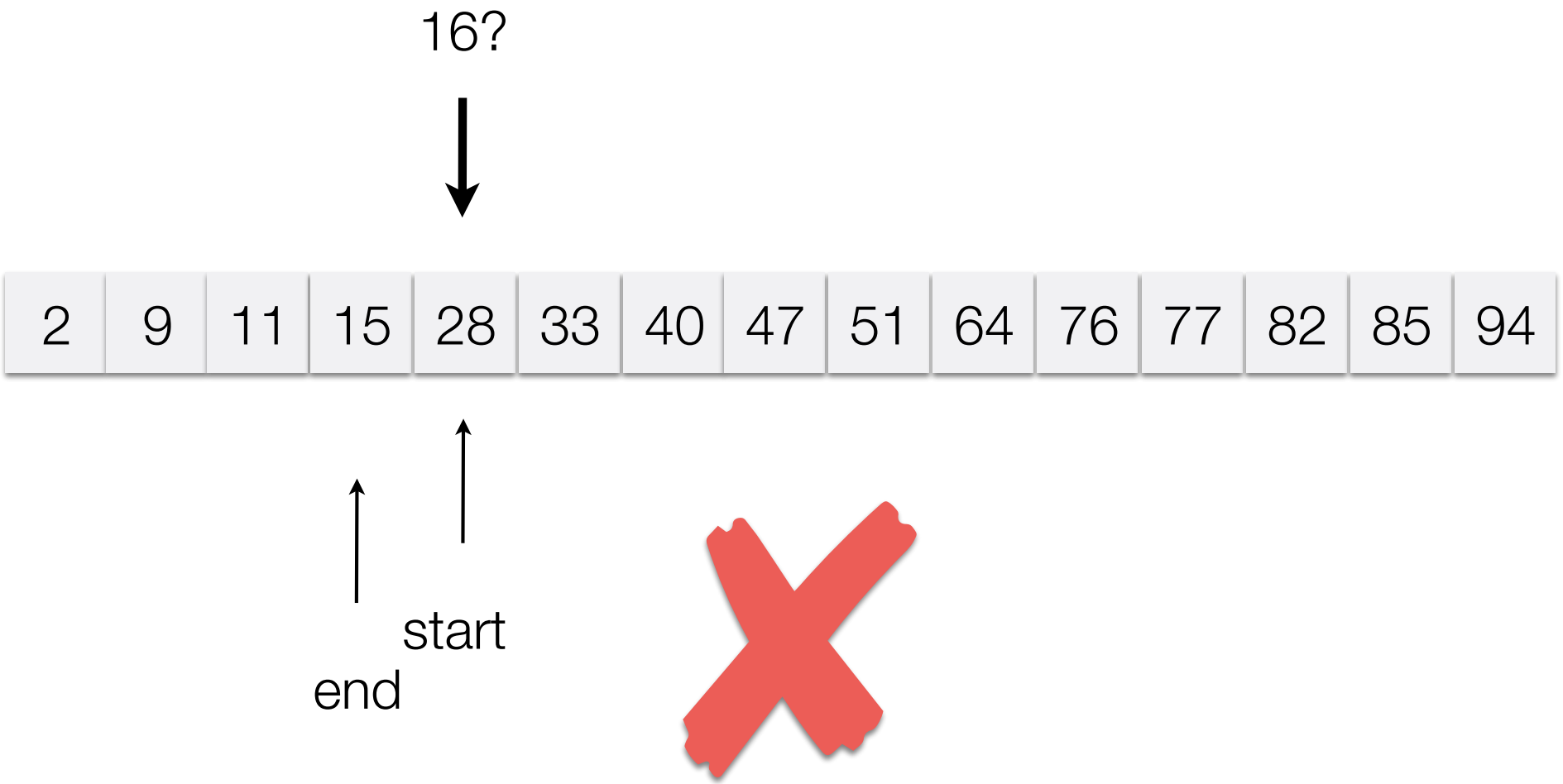
Binary Search



Binary Search



Binary Search



Binary Search

- If array has *n* elements, algorithm needs $\log_2(n)$ steps
- The algorithm is in $\boldsymbol{O(\log_2(n))}$, it is ***logarithmic***

```
static boolean binarySearch(int[] arr, int val, int start, int end) {
  if(start > end) return false;
  int middle = (start + end) / 2;
  if(arr[middle] == val) {
    return true;
  } else if (val < arr[middle]) {
    return binarySearch(arr, val, start, middle - 1);
  } else {
    return binarySearch(arr, val, middle + 1, end);
  }
}
```